



APPENDIX A

```
1: #ifndef __AttrDef_hpp__
2: #define __AttrDef_hpp__
3:
4: #include <iostream.h>
5:
6: ///////////////////////////////////////////////////////////////////
7: ///////////////////////////////////////////////////////////////////
8: ///////////////////////////////////////////////////////////////////
9: ///////////////////////////////////////////////////////////////////
10: //
11: /* Description:
12:  * Attribute (member) definition. Contains RTTI and offset within the object
13:  *
14:  */
15:
16: class RTTI;
17:
18: class AttrDef
19: {
20: public:
21:     /// Create
22:     AttrDef (const RTTI* rtti, int offset);
23:
24:     virtual
25:     ~AttrDef ();
26:
27: public:
28:     /// Use
29:     int
30:     get_offset() const;
31:
32:     const RTTI*
33:     get_rtti() const;
34:
35: public:
36:     /// Printing
37:
38:     friend ostream&
39:     operator<< ( ostream& os, const AttrDef& obj );
40:
41:     // Provide brief description of what is printed.
42:
43: protected:
44:
45:     virtual void
46:     print_on ( ostream& os ) const;
47:
48:     // Called by the output operator.
49:
50: private:
51:     /// Attributes
52:     const RTTI* rtti_;
53:     int offset_;
54:
55: }; // class AttrDef
56:
57: #endif // __AttrDef_hpp__
```

```
1: #include <AttrDef.hpp>
2: #include <RTTI.hpp>
3:
4: ////////////////////////////////////////
5: //                                     AttrDef
6: ////////////////////////////////////////
7:
8: AttrDef::AttrDef (const RTTI* rtti, int offset) :
9:   rtti_(rtti), offset_(offset)
10: {}
11:
12: AttrDef::~AttrDef ()
13: {}
14:
15: int
16: AttrDef::get_offset() const
17: {
18:   return offset_;
19: }
20:
21:
22: const RTTI*
23: AttrDef::get_rtti() const
24: {
25:   return rtti_;
26: }
27:
28: void
29: AttrDef::print_on ( ostream& os ) const
30: {
31:   os << "rtti_ << " \t" << offset_;
32: }
33:
34: ostream&
35: AttrDef::operator<< ( ostream& os, const AttrDef& obj )
36: {
37:   obj.print_on ( os );
38:   return os;
39: }
40: }
```

```
1: #ifndef _AttributeMap_hpp_
2: #define _AttributeMap_hpp_
3:
4: #include <vector>
5: #include <iostream>
6: #include <AttrDef.hpp>
7:
8: ///////////////////////////////////////////////////
9: // AttributeMap
10: ///////////////////////////////////////////////////
11:
12: /*
13: 14: * Abstrust class for Concrete Class Attribute Map (Pointer to Member Map).
15: 15: * Concrete class' AttributeMaps are Singletons
16: 16: *
17: 17: */
18:
19: class AttributeMap
20: {
21: public: // Create
22: virtual
23: ~AttributeMap ();
24:
25: protected: // Create
26: AttributeMap ();
27:
28: public: // Use
29: const AttrDef&
30: getAttrDef (int i) const;
31:
32: int
33: size () const;
34:
35: public: // Printing
36: friend ostream&
37: operator<< ( ostream& os, const AttributeMap& obj );
38: // Provide brief description of what is printed.
39:
40: protected:
41:
42: virtual void
43: print_on ( ostream& os ) const;
44: // Called by the output operator.
45:
46: protected: // Attributes
47: vector <AttrDef, allocator<AttrDef> > attr_map_;
48:
49: }; // class AttributeMap
50:
51: #endif // _AttributeMap_hpp_
52:
```

```
1: #include <AttributeMap.hpp>
2: #include <AttrI.hpp>
3:
4: ////////////////////////////////////////
5: //                                     AttributeMap
6: ////////////////////////////////////////
7:
8: AttributeMap::AttributeMap ()
9: { }
10:
11: AttributeMap::~AttributeMap ()
12: { }
13:
14: const AttrDef&
15: AttributeMap::getAttrDef (int i) const
16: {
17:     return attr_map_[i];
18: }
19:
20: int
21: AttributeMap::size () const
22: {
23:     return attr_map_.size();
24: }
25:
26: void
27: AttributeMap::print_on ( ostream& os ) const
28: {
29:     if (attr_map_.size()) {
30:         os << "Type\tSize\tOffset" << endl;
31:
32:         for (int i = 0; i < attr_map_.size(); i++)
33:             os << attr_map_[i] << endl;
34:     }
35: }
36:
37: ostream&
38: operator<< ( ostream& os, const AttributeMap& obj )
39: {
40:     obj.print_on ( os );
41:     return os;
42: }
```

```
1: #ifndef Externalizer_hpp_
2: #define Externalizer_hpp_
3:
4: #include <iostream>
5:
6: ///////////////////////////////////////////////////////////////////
7: // Externalizer
8: ///////////////////////////////////////////////////////////////////
9:
10: /*
11:  * Description:
12:  * Class to store any object in plain ASCII file
13:  *
14:  */
15:
16: class RTTI;
17:
18: class Externalizer
19: {
20: public:
21:     Externalizer (const RTTI* rtti, ostream& os = cout);
22:
23:     Externalizer ();
24:
25: public:
26:     void
27:     execute(void* ptr);
28:
29: public:
30:     friend ostream&
31:     operator<< ( ostream& os, const Externalizer& obj );
32:     // Provide brief description of what is printed.
33:
34: protected:
35:     virtual void
36:     print_on ( ostream& os ) const;
37:     // Called by the output operator.
38:
39: private:
40:     const RTTI* rtti_;
41:
42:     ostream& os_;
43:
44: private:
45:     void
46:     byte_sequence(char* ptr, int size);
47:
48: }; // class Externalizer
49:
50: #endif // Externalizer_hpp_
51:
```

Externalizer.cpp

1

```

1: #include <Externalizer.hpp>
2: #include <AttributeMap.hpp>
3:
4: #include <RTTI.hpp>
5:
6: ///////////////////////////////////////////////////////////////////
7: // Externalizer
8: ///////////////////////////////////////////////////////////////////
9:
10: Externalizer::Externalizer (const RTTI* rtti, ostream& os) :
11:     rtti_(rtti), os_ (os)
12: {}
13:
14: Externalizer::~Externalizer ()
15: {}
16:
17: void
18: Externalizer::execute ( void* ptr )
19: {
20:     int members = 0;
21:
22:     if (rtti_ -> isBuiltIn()) {
23:         byte_sequence((char*)ptr, rtti_ -> getSize());
24:         os_ << endl;
25:         return;
26:     }
27:
28:     for (int i = 0; i < rtti_ -> attribute_map() -> size(); i++) {
29:         AttrDef def = rtti_ -> attribute_map() -> getAttrDef (i);
30:         const RTTI* r = def.get_rtti ();
31:         void* p = (char*)ptr + def.get_offset();
32:
33:         cout << i << ":-> "; //This output is for illustration
34:
35:         if (r -> isBuiltIn()) {
36:             byte_sequence((char*)p, r -> getSize());
37:             cout << endl; //This output is for illustration
38:             os_ << endl;
39:         } else {
40:             Externalizer ex (r, os_);
41:             ex.execute(p);
42:             return;
43:         }
44:     }
45: }
46:
47: void
48: Externalizer::byte_sequence(char* ptr, int size)
49: {
50:     for (int j = 0; j < size; j++) {
51:         os_ << (unsigned int)(unsigned char)*ptr++ << " ";
52:         cout << (unsigned int)(unsigned char)*(ptr - 1) << " "; //This output
53:         // is for illustration
54:     }
55: }
56:
57: void
58: Externalizer::print_on ( ostream& os ) const
59: {
60:     os << "Externalizer for:\nType\tSize" << endl
61:         << *rtti_;
62: }
63:
64: ostream&
65: Externalizer::operator<< ( ostream& os, const Externalizer& obj )
66: {
67:     obj.print_on ( os );

```

```

68:     return os;
69: }
70:

```

```
1: #ifndef __Internalizer_hpp__
2: #define __Internalizer_hpp__
3:
4: #include <iostream.h>
5:
6: class RTTI;
7:
8: //////////////////////////////////////
9: // Internalizer
10: //////////////////////////////////////
11:
12: /*
13:  * Description:
14:  * Restores an object from an ASCII stream
15:  *
16:  */
17:
18: class Internalizer {
19: public:
20:     // Create
21:     Internalizer (const RTTI* rtti, ostream& is = cin);
22:     virtual ~Internalizer();
23:
24: public:
25:     // Use
26:     void
27:     execute(void* ptr);
28:
29: private:
30:     // Attributes
31:     const RTTI* rtti_;
32:     ostream& is_;
33:
34: private:
35:     // Implementation
36:     void
37:     copy_bytes(char* ptr, int size);
38: };
39:
40: #endif // __Internalizer_hpp__
```



```

1: #include <Internalizer.hpp>
2: #include <AttributeMap.hpp>
3: #include <RTTI.hpp>
4:
5: ///////////////////////////////////////////////////////////////////
6: //                                     Internalizer
7: ///////////////////////////////////////////////////////////////////
8:
9: Internalizer::Internalizer (const RTTI* rtti, istream& is) : rtti_(rtti), is_(
s)
10: {}
11:
12: Internalizer::~Internalizer()
13: {}
14:
15: void
16: Internalizer::execute(void* ptr)
17: {
18:     int members = 0;
19:
20:     if (rtti_ -> isBuiltIn()) {
21:         copy_bytes((char*)ptr, rtti_ -> getSize());
22:         return;
23:     }
24:
25:     for (int i=0; i < rtti_ -> attribute_map() -> size(); i++) {
26:         AttrDef def = rtti_ -> attribute_map() -> getAttrDef(i);
27:         const RTTI* r = def.get_rtti();
28:         void* p = (char*)ptr + def.get_offset();
29:
30:         if (r -> isBuiltIn()) {
31:             copy_bytes((char*)p, r -> getSize());
32:         } else {
33:             Internalizer in (r, is_);
34:             in.execute(p);
35:             return;
36:         }
37:     }
38: }
39:
40: void
41: Internalizer::copy_bytes(char* ptr, int size)
42: {
43:     unsigned int tmp;
44:     for (int j = 0; j < size; j++) {
45:         is_ >> tmp;
46:         *ptr++ = char(tmp);
47:     }
48: }

```

```
1: #ifndef __Object1_hpp_
2: #define __Object1_hpp_
3:
4: #include <iostream.h>
5: #include <Object2.hpp>
6:
7: ///////////////////////////////////////////////////////////////////
8: //                               Object1
9: ///////////////////////////////////////////////////////////////////
10:
11: /*
12:  * Description:
13:  * Test class for Standard Externalization Protocol Development
14:  *
15:  */
16:
17: class AttributeMap;
18:
19: class Object1
20: {
21: public:      /// Create
22:
23:     Object1 ();
24:
25:     Object1 (int n, float f, const Object2& e);
26:
27:     // virtual
28:     ~Object1 ();
29:
30:     /// The following lines are generated by IDL compiler ###
31:     /// The following lines are generated by IDL compiler ###
32:     friend class Object1AttributeMap;
33:
34:     const static AttributeMap*
35:     attribute_map ();
36:
37:     /// The above lines are generated by IDL compiler ###
38:     /// The above lines are generated by IDL compiler ###
39:
40: public:      /// Printing
41:     friend ostream&
42:     operator<< ( ostream& os, const Object1& obj );
43:     // Provide brief description of what is printed.
44:
45: protected:
46:     virtual void
47:     print_on ( ostream& os ) const;
48:     // Called by the output operator.
49:
50: private:    /// Attributes
51:     int a_;
52:     float f_;
53:
54:     Object2 embedded_;
55:
56: }; // class Object1
57:
58: #endif // __Object1_hpp_
59:
```

```

1: #include <Object1.hpp>
2: #include <Object1AttributeMap.hpp>
3:
4: ///////////////////////////////////////////////////////////////////
5: //                                     Object1
6: ///////////////////////////////////////////////////////////////////
7:
8: //const AttributeMap* Object1::attr_map_ptr_ = 0;
9:
10: Object1::Object1 () : a_(0), f_(0), embedded_(0)
11: {}
12:
13: Object1::Object1 (int n, float f, const Object2& e) :
14:   a_(n), f_(f), embedded_(e)
15: {}
16:
17: Object1::~Object1 ()
18: {}
19:
20: // #####
21: // ### The following lines are generated by IDL compiler ###
22: // #####
23: const AttributeMap*
24: Object1::attribute_map ()
25: {
26:     return Object1AttributeMap::map_instance();
27: }
28: // #####
29: // ### The above lines are generated by IDL compiler ###
30: // #####
31:
32: void
33: Object1::print_on ( ostream& os ) const
34: {
35:     os << "0: " << a_ << " ";
36:     << "1: " << f_ << " ";
37:     << "2: " << embedded_;
38: }
39:
40: ostream&
41: operator<< ( ostream& os, const Object1& obj )
42: {
43:     obj.print_on ( os );
44:     return os;
45: }

```

Object1AttributeMap.hpp

```

1: #ifndef __Object1AttributeMap_hpp__
2: #define __Object1AttributeMap_hpp__
3:
4: #include <iostream.h>
5:
6: #include <AttributeMap.hpp>
7:
8: ////////////////////////////////////////
9: //                                     //
10: //                                     //
11: //                                     //
12: //
13: /* Description:
14:  * Map creation and management for Object1 class.
15:  * #####
16:  * ## This class is generated by IDL compiler ##
17:  * #####
18:  */
19:
20: class Object1;
21:
22: class Object1AttributeMap : public AttributeMap
23: {
24: private:
25:     // Create
26:     Object1AttributeMap ();
27:
28: public:
29:     // Create
30:     virtual
31:     ~Object1AttributeMap ();
32:
33:     static const AttributeMap*
34:     map_instance();
35:
36: public:
37:     // Printing
38:     friend ostream&
39:     operator<< ( ostream& os, const Object1AttributeMap& obj );
40:     // Provide brief description of what is printed.
41:
42: protected:
43:     virtual void
44:     print_on ( ostream& os ) const;
45:     // Called by the output operator.
46:
47: private:
48:     static AttributeMap* instance_;
49:     // class Object1AttributeMap
50:
51: #endif // __Object1AttributeMap_hpp__

```

```
1: #include <Object1AttributeMap.hpp>
2: #include <AttributeMap.hpp>
3: #include <Object1.hpp>
4: #include <Object2.hpp>
5: #include <RTTIClassTemplate.hpp>
6: #include <RTTInt.hpp>
7: #include <RTTFloat.hpp>
8:
9: ///////////////////////////////////////////////////////////////////
10: //                               Object1AttributeMap
11: ///////////////////////////////////////////////////////////////////
12:
13: AttributeMap* Object1AttributeMap::instance_ = 0;
14:
15: Object1AttributeMap::Object1AttributeMap ()
16: {
17:     int offset = 0;
18:     int const size = sizeof(Object1);
19:     char tmp[size];
20:
21:     Object1* obj = (Object1*) tmp;
22:
23:     offset = (int)(&(obj->a_)) - (int)(obj);
24:     attr_map_.push_back(AttrDef(new RTTInt (), offset)); // Insert attribute (
member ) definition (a_ is int).
25:
26:     offset = (int)(&(obj->f_)) - (int)(obj);
27:     attr_map_.push_back(AttrDef(new RTTFloat (), offset)); // Insert attribute
( member ) definition (f_ is float).
28:
29:     offset = (int)(&(obj->embedded_)) - (int)(obj);
30:     attr_map_.push_back(AttrDef(new RTTClassTemplate<Object2> ("Object2"), of
set)); // Insert attribute ( member ) definition (embedded_ is Object2).
31: }
32:
33: Object1AttributeMap::~Object1AttributeMap ()
34: {}
35:
36: const AttributeMap*
37: Object1AttributeMap::map_instance()
38: {
39:     if (instance_ == 0) {
40:         instance_ = new Object1AttributeMap ();
41:     }
42:     return instance_;
43: }
44:
45: void
46: Object1AttributeMap::print_on ( ostream& os ) const
47: {
48:     AttributeMap::print_on(os);
49: }
50:
51: ostream&
52: operator<< ( ostream& os, const Object1AttributeMap& obj )
53: {
54:     obj.print_on ( os );
55:     return os;
56: }
```

```
1: #ifndef __Object2_hpp__
2: #define __Object2_hpp__
3:
4: #include <iostream>
5:
6: ////////////////////////////////////////
7: //                                     //
8: //                                     //
9: //                                     //
10: //
11: /*
12:  * Description:
13:  * Test Object2 type
14:  */
15:
16: class AttributeMap;
17:
18: class Object2
19: {
20: public:
21:     /// Create
22:     Object2 ();
23:
24:     Object2 (int n);
25:
26:     virtual
27:     ~Object2 ();
28:     // ##### The following lines are generated by IDL compiler ###
29:     // #####
30:     // #####
31:     friend class Object2AttributeMap;
32:
33:     const static AttributeMap*
34:     attribute_map ();
35:     // #####
36:     // ##### The above lines are generated by IDL compiler ###
37:     // #####
38:
39: public:
40:     /// Use
41:     void
42:     set (int n);
43:
44:     int
45:     get ();
46:
47: public:
48:     /// Printing
49:     friend ostream&
50:     operator<< (ostream& os, const Object2& obj );
51:     // Provide brief description of what is printed.
52:
53: protected:
54:     virtual void
55:     print_on ( ostream& os ) const;
56:     // Called by the output operator.
57:
58: private:
59:     /// Attributes
60:     int n_;
61:
62: }; // class Object2
63:
64: #endif // __Object2_hpp__
```

```
1: #include <Object2.hpp>
2: #include <Object2AttributeMap.hpp>
3:
4: ///////////////////////////////////////////////////////////////////
5: //                                     Object2
6: ///////////////////////////////////////////////////////////////////
7:
8: Object2::Object2 ( ) : n_(0)
9: {
10:
11: Object2::Object2 (int n) : n_(n)
12: {
13:
14: Object2::~Object2 ( )
15: {
16:
17: // #####
18: // ### The following lines are generated by IDL compiler ###
19: // #####
20: const AttributeMap*
21: Object2::attribute_map ( )
22: {
23:     return Object2AttributeMap::map_instance();
24: }
25: // #####
26: // ### The above lines are generated by IDL compiler ###
27: // #####
28:
29: void
30: Object2::set (int n)
31: {
32:     n_ = n;
33: }
34:
35: int
36: Object2::get ( )
37: {
38:     return n_;
39: }
40:
41: void
42: Object2::print_on ( ostream& os ) const
43: {
44:     os << "0: " << n_;
45: }
46:
47: ostream&
48: operator<< ( ostream& os, const Object2& obj )
49: {
50:     obj.print_on ( os );
51:     return os;
52: }
```

Object2AttributeMap.hpp

```

1: #ifndef __Object2AttributeMap_hpp__
2: #define __Object2AttributeMap_hpp__
3:
4: #include <iostream.h>
5:
6: #include <AttributeMap.hpp>
7:
8: ////////////////////////////////////////
9: //                                     Object2AttributeMap
10: ////////////////////////////////////////
11:
12: /*
13:  * Description:
14:  * Map creation and management for Object2 class.
15:  * #####
16:  * ### This class is generated by IDL compiler ###
17:  * #####
18:  */
19:
20: class Object2;
21:
22:
23: class Object2AttributeMap : public AttributeMap
24: {
25: private:
26:     Object2AttributeMap ();
27:
28: public:
29:     virtual
30:     ~Object2AttributeMap ();
31:
32:     static const AttributeMap*
33:     map_instance();
34:
35: public:
36:     /// Printing
37:
38:     friend ostream&
39:     operator<< ( ostream& os, const Object2AttributeMap& obj );
40:     // Provide brief description of what is printed.
41:
42: protected:
43:
44:     virtual void
45:     print_on ( ostream& os ) const;
46:     // Called by the output operator.
47:
48: private:
49:     static AttributeMap* instance_;
50: }; // class Object2AttributeMap
51:
52:
53: #endif // __Object2AttributeMap_hpp__
54:

```



```
1: #include <Object2AttributeMap.hpp>
2: #include <AttributeMap.hpp>
3: #include <Object2.hpp>
4: #include <RTTIClassTemplate.hpp>
5: #include <RTTIInt.hpp>
6:
7: ///////////////////////////////////////////////////////////////////
8: //                               Object2AttributeMap
9: ///////////////////////////////////////////////////////////////////
10:
11: AttributeMap* Object2AttributeMap::instance_ = 0;
12:
13: Object2AttributeMap::Object2AttributeMap ()
14: {
15:     int offset = 0;
16:     int const size = sizeof(Object2);
17:     char tmp[size];
18:
19:     Object2* obj = (Object2*) tmp;
20:
21:     offset = (int)(&(obj->n_)) - (int)obj;
22:     attr_map_.push_back(AttrDef(new RTTIInt (), offset)); // Insert attribute (
member ) definition (a_ is int).
23: }
24:
25: Object2AttributeMap::~Object2AttributeMap ()
26: {}
27:
28: const AttributeMap*
29: Object2AttributeMap::map_instance()
30: {
31:     if (instance_ == 0) {
32:         instance_ = new Object2AttributeMap ();
33:     }
34:     return instance_;
35: }
36:
37: void
38: Object2AttributeMap::print_on ( ostream& os ) const
39: {
40:     AttributeMap::print_on(os);
41: }
42:
43: ostream&
44: operator<< ( ostream& os, const Object2AttributeMap& obj )
45: {
46:     obj.print_on ( os );
47:     return os;
48: }
```

```
1: #ifndef __RTTI_hpp_
2: #define __RTTI_hpp_
3:
4: #include <iostream.h>
5:
6: ///////////////////////////////////////////////////////////////////
7: // RTTI
8: ///////////////////////////////////////////////////////////////////
9:
10: class AttributeMap;
11:
12: /*
13:  * Description:
14:  * Abstract class for RTTI
15:  *
16:  */
17:
18: #include <string>
19:
20: class RTTI
21: {
22: public:
23:     RTTI (const char* t, bool builtin);
24:
25:     virtual
26:     ~RTTI ();
27:
28: public:
29:     /// Configuration
30:     virtual int
31:     getSize() const = 0;
32:
33:     virtual const AttributeMap*
34:     attribute_map () const = 0;
35:
36:     bool
37:     isBuiltin() const;
38:
39:     const string&
40:     getType() const;
41:
42: public:
43:     /// Printing
44:     friend ostream&
45:     operator<< ( ostream& os, const RTTI& obj );
46:
47:     // Provide brief description of what is printed.
48:
49: protected:
50:
51:     virtual void
52:     print_on ( ostream& os ) const;
53:
54:     // Called by the output operator.
55:
56: private:
57:     /// Attributes
58:     bool builtin_;
59:     // It's 'true' for built-in types (int, float, double, char) and 'false' f
60:     // user defined (classes, structures, unions)
61:     string type_;
```

```
1: #include <RTTI.hpp>
2:
3: ////////////////////////////////////////
4: // RTTI
5: ////////////////////////////////////////
6:
7: RTTI::RTTI (const char* t, bool builtin) : type_(string(t)), builtin_(builtin)
8: {}
9:
10: RTTI::~RTTI ()
11: {}
12:
13: const strings
14: RTTI::getType() const
15: {
16:     return type_;
17: }
18:
19: bool
20: RTTI::isBuiltin() const
21: {
22:     return builtin_;
23: }
24:
25: void
26: RTTI::print_on ( ostream& os ) const
27: {
28:     os << type_ << "\t" << getSize();
29: }
30:
31: ostream&
32: operator<< ( ostream& os, const RTTI& obj )
33: {
34:     obj.print_on ( os );
35:     return os;
36: }
```

```
1: #ifndef __RTTIClassTemplate_hpp__
2: #define __RTTIClassTemplate_hpp__
3:
4:
5: #include <iostream.h>
6:
7: #include <RTTI.hpp>
8:
9: ///////////////////////////////////////////////////////////////////
10: // RTTIClassTemplate<T>
11: ///////////////////////////////////////////////////////////////////
12:
13: /*
14:  * Description:
15:  * Class template to create concrete RTTI class objects
16:  *
17:  */
18:
19: class AttributeMap;
20:
21: template <class T>
22: class RTTIClassTemplate : public RTTI
23: {
24: public:
25:     RTTIClassTemplate (const char* t);
26:
27:     virtual
28:     ~RTTIClassTemplate ();
29:
30: public:
31:     virtual int
32:     getSize() const;
33:
34:     virtual const AttributeMap*
35:     attribute_map () const;
36:
37: public:
38:     friend ostream&
39:     operator<< ( ostream& os, const RTTIClassTemplate<T>& obj );
40:     // Provide brief description of what is printed.
41:
42: protected:
43:     virtual void
44:     print_on ( ostream& os ) const;
45:     // Called by the output operator.
46:
47: }; // class RTTIClassTemplate<T>
48:
49: #ifndef NO_COMPILE_INSTANTIATE
50: #include <RTTIClassTemplate.cc>
51: #endif
52:
53: #endif // __RTTIClassTemplate_hpp__
54:
```

```
1: #include <RTTIClassTemplate.hpp>
2: #include <AttributeMap.hpp>
3:
4: ///////////////////////////////////////////////////////////////////
5: // RTTIClassTemplate<T>
6: ///////////////////////////////////////////////////////////////////
7:
8: template <class T>
9: RTTIClassTemplate<T>::RTTIClassTemplate (const char* t) :
10:     RTTI(t, false)
11: {}
12:
13: template <class T>
14: RTTIClassTemplate<T>::~RTTIClassTemplate ()
15: {}
16:
17: template <class T>
18: int
19: RTTIClassTemplate<T>::getSize() const
20: {
21:     return (sizeof (T));
22: }
23:
24: template <class T>
25: const AttributeMap*
26: RTTIClassTemplate<T>::attribute_map () const
27: {
28:     return T::attribute_map();
29: }
30:
31: template <class T>
32: void
33: RTTIClassTemplate<T>::print_on ( ostream& os ) const
34: {
35:     RTTI::print_on(os);
36: }
37:
38: template <class T>
39: ostream&
40: operator<< ( ostream& os, const RTTIClassTemplate<T>& obj )
41: {
42:     obj.print_on ( os );
43:     return os;
44: }
```

```
1: #ifndef __RTTifloat_hpp__
2: #define __RTTifloat_hpp__
3:
4: #include <iostream.h>
5:
6: #include <RTTI.hpp>
7:
8: ////////////////////////////////////// RTTifloat
9: //////////////////////////////////////
10: //////////////////////////////////////
11:
12: /*
13:  * Description:
14:  * RTTI for built-in 'float' type
15:  *
16:  */
17:
18: class RTTifloat : public RTTI
19: {
20: public:      /// Create
21:
22:     RTTifloat ();
23:
24:     virtual
25:     ~RTTifloat ();
26:
27:
28: public:      /// Use
29:     virtual int
30:     getSize() const;
31:
32:     virtual const AttributeMap*
33:     attribute_map () const;
34:
35: public:      /// Printing
36:     friend ostream&
37:     operator<< ( ostream& os, const RTTifloat& obj );
38:     /// Provide brief description of what is printed.
39:
40: protected:
41:     virtual void
42:     print_on ( ostream& os ) const;
43:     /// Called by the output operator.
44:
45: }; // class RTTifloat
46:
47:
48: #endif // __RTTifloat_hpp__
49:
```

```
1: #include <RTTifloat.hpp>
2:
3: //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
4: //                                                                 RTTifloat
5: //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
6:
7:
8: RTTifloat::RTTifloat () : RTTI("float", true)
9: {}
10:
11: RTTifloat::~RTTifloat ()
12: {}
13:
14: int
15: RTTifloat::getSize() const
16: {
17:     return (sizeof (float));
18: }
19:
20: const AttributeMap*
21: RTTifloat::attribute_map () const
22: {
23:     throw ("Built-in 'float' type has no attribute map");
24: }
25:
26: void
27: RTTifloat::print_on ( ostream& os ) const
28: {
29:     RTTI::print_on(os);
30: }
31:
32: ostream&
33: operator<< ( ostream& os, const RTTifloat& obj )
34: {
35:     obj.print_on ( os );
36:     return os;
37: }
```

```
1: #ifndef __RTTIInt_hpp_
2: #define __RTTIInt_hpp_
3:
4: #include <iostream.h>
5:
6: #include <RTTI.hpp>
7:
8: //////////////////////////////////////
9: // RTTIInt
10: //////////////////////////////////////
11:
12: /*
13:  * Description:
14:  * RTTI for built-in 'int' type
15:  *
16:  */
17:
18: class RTTIInt : public RTTI
19: {
20: public:
21:     /// Create
22:     RTTIInt ();
23:
24:     virtual
25:     ~RTTIInt ();
26:
27:
28: public:
29:     /// Use
30:     virtual int
31:     getSize() const;
32:
33:     virtual const AttributeMap*
34:     attribute_map () const;
35:
36: public:
37:     /// Printing
38:     friend ostream&
39:     operator<< ( ostream& os, const RTTIInt& obj );
40:     // Provide brief description of what is printed.
41:
42: protected:
43:     virtual void
44:     print_on ( ostream& os ) const;
45:     // Called by the output operator.
46:
47: }; // class RTTIInt
48:
49: #endif // __RTTIInt_hpp_
```



```
1: #include <RTTIInt.hpp>
2:
3: ///////////////////////////////////////////////////////////////////
4: // RTTIInt
5: ///////////////////////////////////////////////////////////////////
6:
7:
8: RTTIInt::RTTIInt () : RTTI("int", true)
9: {}
10:
11: RTTIInt::~RTTIInt ()
12: {}
13:
14: int
15: RTTIInt::getSize() const
16: {
17:     return (sizeof (int));
18: }
19:
20: const AttributeMap*
21: RTTIInt::attribute_map () const
22: {
23:     throw ("Built-in 'int' type has no attribute map");
24: }
25:
26: void
27: RTTIInt::print_on ( ostream& os ) const
28: {
29:     RTTI::print_on(os);
30: }
31:
32: ostream&
33: operator<< ( ostream& os, const RTTIInt& obj )
34: {
35:     obj.print_on ( os );
36:     return os;
37: }
38:
```

```

1: #include <iostream.h>
2: #include <sstream.h>
3:
4: #include <Object1.hpp>
5: #include <Externalizer.hpp>
6: #include <Internalizer.hpp>
7: #include <RTTIClassTemplate.hpp>
8: #include <RTTIfloat.hpp>
9:
10: int
11: main ()
12: {
13:     cout << endl;
14:     cout << "Creating an object of user defined Object1 type object" << endl;
15:
16:     Object1 obj(3, 7, 21);
17:     const AttributeMap* map = Object1::attribute_map();
18:
19:     cout << "\nObject1: " << obj << endl;
20:
21:     cout << "\nAttribute Map of the Object1: " << endl;
22:     cout << *map << endl;
23:
24:     cout << "Creating an Externalizer for a given type (Object1)" << endl;
25:
26:     char temp[1024];
27:     ostream ostr(temp, sizeof(temp));
28:
29:     Externalizer ext_obj (new RTTIClassTemplate<Object1> ("Object1"), ostr);
30:     cout << ext_obj << endl;
31:
32:     cout << "\nExternalizing raw memory that contains an object of given type
Object1 type object)" << endl;
33:     ext_obj.execute((void*) &obj);
34:
35:     cout << "Done" << endl;
36:
37:     cout << "\nCreating an Internalizer for a given type (Object1)" << endl;
38:
39:     istream istr(temp, ostr.pcount());
40:
41:     Internalizer int_obj(new RTTIClassTemplate<Object1> ("Object1"), istr);
42:     Object1 obj_result;
43:
44:     cout << "Internalizing from raw memory into Object1" << endl;
45:     int_obj.execute((void*) &obj_result);
46:     cout << "Result " << obj_result << endl;
47:
48:     cout << "\nCreating an object float f = 3" << endl;
49:
50:     float f = 3;
51:
52:     cout << "\nCreating an Externalizer for a given type (float)" << endl;
53:     char temp2[1024];
54:     ostream ostr2(temp2, sizeof(temp2));
55:
56:     Externalizer ext_f (new RTTIfloat, ostr2);
57:     cout << ext_f << endl;
58:
59:     cout << "\nExternalizing raw memory that contains an object of given type
float type object)" << endl;
60:     ext_f.execute((void*) &f);
61:     cout << "\nDone" << endl;
62:
63:     cout << "\nCreating an Internalizer for a given type (float)" << endl;
64:     istream istr2(temp2, ostr2.pcount());
65:     Internalizer int_f(new RTTIfloat, istr2);
66:     float fit;
67:
68:     cout << "Internalizing from raw memory into float" << endl;
69:     int_f.execute((void*) &fit);
70:     cout << "Result " << fit << endl;
71:
72:     return 0;
73: }

```

PRINTOUTS.TXT

Creating an object of user defined Object1 type object

Object1: 0: 3; 1: 7; 2: 0: 21

Attribute Map of the Object1:

Type	Size	Offset
------	------	--------

int	4	0
-----	---	---

float	4	4
-------	---	---

Object2	8	8
---------	---	---

Creating an Externalizer for a given type (Object1)

Externalizer for:

Type	Size
------	------

Object1	20
---------	----

Externalizing raw memory that contains an object of given type (Object1 type object)

0:-> 0 0 0 3

1:-> 64 224 0 0

2:-> 0:-> 0 0 0 21

Done

Creating an Internalizer for a given type (Object1)

Internalizing from raw memory into Object1

Result 0: 3; 1: 7; 2: 0: 21

Creating an object float f = 3

Creating an Externalizer for a given type (float)

Externalizer for:

Type	Size
------	------

float	4
-------	---

Externalizing raw memory that contains an object of given type (float type object)

64 64 0 0

Done

Creating an Internalizer for a given type (float)

Internalizing from raw memory into float

Result 3